Database - Feature #4030

improve CompoundQuery optimizer

04/01/2019 10:36 PM - Eric Faulhaber

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Database - Bug #4917: eliminate redundant ORDER BY elements in mul...	**New**

History

#1 - 04/01/2019 10:52 PM - Eric Faulhaber

Some time ago, I implemented a runtime optimizer for CompoundQuery, which would analyze the components of the compound query and attempt to synthesize a database server-side join to make the query more efficient.

The default behavior of CompoundQuery is to join its multiple tables at the FWD application server. This means it executes N + 1 selects at each pair of joined tables, which is generally an expensive proposition. The optimizer seeks to identify cases where these joins can be safely moved to the database server, to execute a single select to replace those default, N + 1 selects.

The results of the initial implementation are mixed. Some queries improved dramatically (like, minutes to seconds). Others actually got worse, due to the server-side join resulting in a more expensive overall plan than if we had just allowed the default behavior to continue.

As a result of testing the optimizer with a lot of compound queries, I ended up watering down the implementation to a much less aggressive joining algorithm. Now, we essentially just optimize joins of very limited complexity and allow the rest to continue un-optimized.

I think there is still headroom for improvement here. The initial testing was done with a limited variety of compound queries against PostgreSQL 9.1. We probably need to attack this problem again with a greater variety of queries and a newer version of PostgreSQL. It may be that we have to make the optimizer dialect-specific as well, since database query planners will differ in the plans they produce for various joins. Also, I did no query planner tuning at all as part of the initial effort, and we know that at least in PostgreSQL, the tuning parameters can make a big difference in the final query plans.

Another thing to consider is how to effectively and efficiently cache the optimizer's decisions, so we can short-circuit the optimization decision tree when we hit what is essentially the same type of compound query we've encountered before.

One thing holding us back here is that to some degree we are flying blind in our optimization decisions, in that we don't have access to any database statistics as an input.

#2 - 09/27/2020 03:44 PM - Greg Shah

- Related to Bug #4917: eliminate redundant ORDER BY elements in multi-table queries added